Problem 1: The value computed by the program below approaches $\pi$. Re-write the program in MIPS assembler. The code should execute quickly. Assume that all integer instructions take one cycle, floating-point adds take four cycles and divides take ten cycles. Make changes to the code to improve speed (possibly using an integer for $i$ or even using both an integer and double). Do not use a different technique for computing $\ldots$.

```c
int main(int argv, char **argc)
{
    double i;
    double sum = 0;

    for(i=1; i<50000000;)
    {
        sum = sum + 4.0 / i; i += 2;
        sum = sum - 4.0 / i; i += 2;
    }

    printf("After %d iterations sum = \%.8f\n", (int)(i-1)/2, sum);

    return 0;
}
```

Problem 2: The program below is used to generate a password based on the outcome of several rolls of a twenty-sided die. The program was compiled using the Sun Workshop Compiler 5.0 targeting SPARC V7 ($-xarch=v7$) and SPARC V9 ($-xarch=v8plus$, code which can run on a V9 processor with a 32-bit OS), the output of the compiler is shown for the for loop.


Here are a few useful facts about SPARC:

- Register names for SPARC are: $%g0-%g7$ (global), $%l0-%l7$ (local), $%i0-%i7$ (input), $%o0-%o7$ (output), and $%f0-%f31$ (floating point). Registers $%fp$ (frame pointer) and $%sp$ are aliases for $%i6$ and $%o6$, respectively. Register $%g0$ is a zero register.

- Local variables (the only kind used in the code fragment shown) are stored in memory at some offset from the stack pointer (in $%sp$). For example, `1dd $[%sp+96],%f0` loads a local variable into register $%f0$.

- All V7 and V8 integer registers are 32 bits. V9 registers are 64 bits but with the v8plus option only the 32 lower bits are used.

- Unlike MIPS and DLX, the last register in an assembly language instruction is the destination. For example, `add $%g1, $%g2, $%g3`, puts the sum of $g1$ and $g2$ in register $g3$.

Like MIPS, SPARC branches are delayed. Unlike MIPS, some delayed branches are annulled, indicated with a “\(a\)” in the mnemonic. In an annulled branch the instruction in the delay slot is executed if and only if the branch is taken.

(a) For each compilation, identify which registers are used for which program variables.
(b) For each instruction used in the V9 version of the code but not in the V7 version, explain what it does and how it improves execution over the V7 version.

```c
int main(argc, argv)
    int argc;
    char **argv;
{
   int die_rolls[] = {15, 17, 6, 10, 19, 19, 15, 17, 16, 5, 0 };  
   int *rolls_ptr = &die_rolls[0];
   char pw[8];
   char *pw_ptr = &pw[0];

   int faces_per_die = 20; /* Available at Little Wars in Village Square */
   double bits_per_roll = log(faces_per_die)/log(2.0);
   double bits_per_letter = log(26.0) / log(2.0);

   double bits = 0.0;
   uint64_t seed = 0;    /* A 64-bit integer. */
   int roll;

   while( ( roll = *rolls_ptr++ ) )
   {
      seed = faces_per_die * seed + (roll-1);
      bits += bits_per_roll;
   }

   for( ; bits >= bits_per_letter; bits -= bits_per_letter )
   {
      *pw_ptr++ = 'a' + seed % 26;
      seed = seed / 26;
   }
   *pw_ptr = 0;

   printf("The password is %s\n",pw);
   return 0;
}
```

! Compiled with -xarch=v7
!
! 32 ! for( ; bits >= bits_per_letter; bits -= bits_per_letter )

/* 0x010c  32 */ ldd [%sp+96],%f0
.L900000118:
/* 0x0110  32 */ fcmped %f30,%f0
/* 0x0114  */ nop
/* 0x0118  */ fbul .L77000009
/* 0x011c  */ or %g0,0,%o2
   .L900000116:
for( ; bits >= bits_per_letter; bits -= bits_per_letter )


/* 0x0100  35 */ or %g0,%g2,%o0
/* 0x0104 */ fsubd %f8,%f4,%f8
/* 0x0108  34 */ add %g3,97,%g3
/* 0x010c */ stb %g3,[%i0]
/* 0x0110 */ add %i0,1,%i0
/* 0x0114  35 */ fcmped %fcc1,%f8,%f4
/* 0x0118 */ fbge,a,pt %fcc1,.L900000114
/* 0x011c */ udivx %o0,26,%g2
  .L77000009:

! 36 !  
}